Regex en Java: Validando datos con expresiones regulares

alex-felipe
13/08/2021

En el menú sistema de registro de alumnos recibo un alumno con los siguientes atributos:

```
public class Alumno {

private String nombre;
private String apellido;
private String telefono;

//métodos

}
```

Al imprimir los datos de un alumno:

```
Nombre: Alex2016 Apellido: 12Felipe Teléfono: 11992232121455
```

¿Alex2016? *¿12 Felipe*? ¿Y este extraño teléfono? Este alumno contiene informaciones que no tienen sentido...

¿Cómo puedo evitar que se ingresen estos tipos de datos en mi sistema?

Intentemos validar estos datos creando un método que valide al alumno:

```
public boolean valida(Alumno alumno) {

//implementación

}
```

Comencemos con el nombre del alumno. ¡No queremos que tenga números al principio o al final del nombre! ¿**Cómo** podemos hacer esto?

Extraigamos tanto el primer carácter como el último a través del método `charAt()` de la clase `String`:

```
public boolean valida(Alumno alumno) {

String nombre = alumno.getNombre();
char primeraLetra = nombre.charAt(0);
char ultimaLetra = nombre.charAt(nombre.length() - 1)

}
```

Para **comprobar si un carácter es una letra**, podemos usar el método estático `isAlphabetic()` de la clase Character:

```
public boolean valida(Alumno alumno) {

String nombre = alumno.getNombre();

if (Character.isAlphabetic((nombre.charAt(0))) &&
Character.isAlphabetic((nombre.charAt(nombre.length() - 1)))) {
    return true;
    }

return false; }
```

¡Implementamos nuestra primera validación! Ahora agreguemos un `sysout` dentro de un `if` para que probemos:

```
if (valida(alumno)) {
    System.out.println("alumno " + alumno.getNombre() + " es válido");
    } else {
        System.out.println("alumno " + alumno.getNombre() + " es inválido");
}
```

Probemos un nombre que comience con un número, en este caso "1Alex":

```
alumno 1Alex es inválido
```

¡Excelente! Funcionó como se esperaba. Pero, ¿y si el número está al final? Por ejemplo, lo que se presentó al principio: "Alex2016". Veamos el resultado:

```
alumno Alex2016 no es válido
```

¡Aparentemente todo está funcionando como se esperaba! Ahora, ¿qué pasa si hay un número en el medio del nombre? Por ejemplo, "A1ex":

```
alumno A1ex es válido
```

¡Uf! ¡No pensamos en ese caso! ¿Cómo podemos hacer que el nombre del alumno no tenga ningún número? Tendremos que "barrer" toda nuestra String, carácter a carácter y comprobar si es válido:

```
public boolean valida(Alumno alumno) {

String nombre = alumno.getNombre();

for (int i = 0; i < nombre.length(); i++) {
    if (!Character.isAlphabetic((nombre.charAt(i)))) {
        return false; } }
```

```
return true; }
```

Si probamos de nuevo con el nombre "A1ex":

```
alumno A1ex no es válido
```

¡Excelente! ¡Ahora el nombre de mi alumno está siendo validado como se esperaba! ¿Qué pasa si recibimos a un alumno que ha ingresado un nombre con una sola letra? Como "A" por ejemplo ... ¿qué pasaría?

```
alumno A es válido
```

Hmmm, pero seguramente para mí sistema no tiene sentido tener un alumno con un nombre de solo 1 letra... así que agregaremos una condición más en nuestro validador:

```
public boolean valida(Alumno alumno) {

String nombre = alumno.getNombre();

if(nombre.length() > 2){ return false; }

*//restante del código*

}
```

Si probamos ahora:

```
alumno A no es válido
```

Nuestra validación está funcionando por ahora, pero mira el tamaño del método:

```
public boolean valida(Alumno alumno) {

String nombre = alumno.getNombre();
```

```
if(nombre.length() > 2){ return false; }

for (int i = 0; i > nombre.length(); i++) {
    if (!Character.isAlphabetic((nombre.charAt(i)))) {
        return false; } }

return true; }
```

A primera vista, ¿podemos entender lo que se está haciendo? Probablemente no.

También ten en cuenta que **en cada validación** que necesitamos hacer son más `ifs` que necesitamos agregar. ¿Existe otra forma más sencilla de validar estos datos? ¡La respuesta es **sí**!

Podemos usar **expresiones regulares** a través del método `matches()` de la clase `String`. Definamos nuestra expresión regular para validar el nombre del alumno:

```
public boolean valida(Alumno alumno) {

String nombre = alumno.getNombre();

return nombre.matches("\[a-z\]");

}
```

De acuerdo, esta expresión regular significa que esperamos **solo 1 única letra del alfabeto**. Pero lo que queremos es que tenga al menos 2, ¿verdad? ¡Sencillo! Solo agrega llaves ({}) e informa el número de repeticiones que desea:

```
public boolean valida(Alumno alumno) {

String nombre = alumno.getNombre();

return nombre.matches("\[a-z\]{2}");
```

```
}
```

En este ejemplo estamos diciendo que queremos **solo** 2 repeticiones, es decir, la cantidad **no puede ser más grande o más pequeña ¡**es si **igual a 2 letras**!

¡Definitivamente esto no es lo que queremos! ¿Y ahora? ¿Cómo podemos decir que queremos al menos 2 letras?

Cuando usamos llaves podemos informar 2 parámetros: el primero significa la **cantidad mínima** y el segundo la **cantidad máxima**. Es decir, si agregamos solo una ",":

```
public boolean valida(Alumno alumno) {

String nombre = alumno.getNombre();

return nombre.matches("\[a-z\]{2,}");

}
```

Ahora mismo estamos diciendo que ¡esperamos al **menos 2 letras** y **la cantidad máxima no tiene límite**!Si probamos el nombre "Alex":

```
alumno Alex no es válido
```

¿Qué sucedió? ¿No se suponía que era válido? Nuestra expresión regular dice que espera al menos 2 letras, sin embargo ¡**letras minúsculas**! Ahora bien, ¿cómo podemos hacer que nuestra expresión regular tenga una letra mayúscula al principio? De la misma manera que para minúsculas usamos [a-z] nosotros podemos usar [A-Z] para letras mayúsculas:

```
public boolean valida(Alumno alumno) {

String nombre = alumno.getNombre();

return nombre.matches("\[A-Z\]\[a-z\]{1,}");
```

```
}
```

Ahora nuestra expresión regular espera una String con la primera letra en mayúscula, ¡por lo que solo debemos asegurarnos de que tenga al menos 1 letra minúscula después! Si intentamos ejecutar de nuevo:

```
alumno alex es válido
```

¿Qué pasa si intentamos con esos nombres inválidos? Por ejemplo: ¿"A1ex", "1Alex" y "Alex2016"? Veamos el resultado:

```
alumno A1ex no es válido
alumno 1Alex no es válido
alumno Alex2016 no es válido
```

¡Estupendo! ¡Todo funciona como se esperaba y de una manera más clara y fácil de mantener! si un día necesitamos **modificar nuestra validación**, simplemente adapte nuestra expresión regular a la nueva regla de negocio :)

¿Qué te pareció la expresión regular? ¿Qué tal un desafío? Define una expresión regular para **validar un teléfono** que solo acepta teléfonos en estos 2 formatos: (11) 1111-1111 o (11) 11111-1111. ¡Cuéntanos tu resultado!

Entonces, ¿te gustaron las expresiones regulares? ¿Qué tal aprender más sobre el tema? **En Alura Latam, ¡el curso de expresión regular aborda más detalles sobre este poderoso recurso que hace la vida mucho más fácil para el programador al realizar varias validaciones!**

Puedes leer también:

- Entidades Managed, Transient y Detached en Hibernate y JPA
- Entendiendo el Lazy y el Eager Load de JPA
- JPA con Hibernate: Herencia y Mapeos

ARTÍCULOS DE TECNOLOGÍA > PROGRAMACIÓN

¡QUIERO EMPEZAR A ESTUDIAR!

Paga en moneda local en los siguientes países

ANUAL

US$79,90

un solo pago de US$79,90

- 218 cursos
- Videos y actividades 100% en Español
- Certificado de participación
- Estudia las 24 horas, los 7 días de la semana
- Foro y comunidad exclusiva para resolver tus dudas
- Acceso a todo el contenido de la plataforma por 12 meses

¡QUIERO EMPEZAR A ESTUDIAR!

BLOG

PROGRAMACIÓN
FRONT END
DATA SCIENCE
INNOVACIÓN Y GESTIÓN
DEVOPS

AOVS Sistemas de Informática S.A
CNPJ 05.555.382/0001-33

SÍGUENOS EN NUESTRAS REDES SOCIALES

ALIADOS

Empresa participante do SCALE UP ENDEAVOR

En Alura somos unas de las Scale-Ups seleccionadas por Endeavor, programa de aceleración de las empresas que más crecen en el país.

Google for Startups Alumni 2021

Fuimos unas de las 7 startups seleccionadas por Google For Startups en participar del programa Growth Academy en 2021

POWERED BY

CURSOS

Cursos de Programación

Lógica de Programación | Java

Cursos de Front End

HTML y CSS | JavaScript | React

Cursos de Data Science

Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

Cursos de DevOps

Docker | Linux

Cursos de Innovación y Gestión

Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics | Liderazgo y Gestión de Equipos | Startups y Emprendimiento